



8911 North Capital of Texas Highway
Austin, Texas
+1.512.343.1010
www.liant.com

SOA and COBOL

By John Bradley, President and CEO

Contents

Introduction	2
SOA and Its Importance	3
The Choices for COBOL, Until Now	3
The Xcentricity™ Solution	4
How to Implement an SOA Solution	5
Summary	8

Introduction

One would have to be a hermit not to have been exposed to the hype associated with an approach to IT application design and construction known as *Service-Oriented Architecture*, or simply *SOA*. One is just as likely to be confused by the seemingly universal application of this term to every vendor's latest products, especially those that complement and used worldwide Web technology and infrastructure. Just to set the stage and, hopefully, separate this from some of the less content-rich marketing documents on this topic, let's start by defining what we are talking about.

In its reference model for SOA, the Organization for the Advancement of Structured Information Standards (OASIS) defines SOA to be "... a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains."ⁱ This definition seems to be very broad and unhelpful, until you examine it in detail.

First, it defines SOA as a "paradigm" or model, not an implementation or technique. This means that SOA guides the building of IT solutions, but it doesn't itself solve any of the problems. Instead it encourages, if not dictates, the use of techniques and strategies that do indeed solve problems.

Second, it declares that it offers guidance as to organization and utilization of disparate resources. These resources can be presumed to be any and all IT artifacts, from data files and databases to complete applications, and everything in between.

Finally, and most importantly, it disallows the presumption that all of the resources are "owned" by the same organization. This condition is crucial, for virtually all IT

solutions built over the past 40 years have, in fact, made the assumption either explicitly or implicitly that all of the components of the solution are built and controlled by the same organization. So-called application-level integration technologies such as electronic data interchange (EDI) attempt to mitigate the consequences of this legacy design assumption, but only a fundamental architectural embracing of the concept of multiple domains of control can hope to produce a solution that can survive and thrive in the reality of today's IT environment. SOA is the expression of that concept.

SOA and Its Importance

The problem that SOA attempts to solve is that of the reality of control. As programs comprise applications, applications comprise solutions, and solutions comprise IT systems, the odds of a modern IT system being composed of single-source components and data is nil. If we don't expect this and design and build our applications to deal with it, we are doomed to be left behind by those who do. While it might seem like SOA is "just another way to build an application," it is really the *only* way to build a modern application, particularly if the application is to have any chance at all of being useful for more than a few years.

Central to SOA is the concept of a "service." A service implies that something is doing work for, or on the behalf of, something else. For this to take place, the work being offered must be useful to the client,

the server must be willing to perform it, and the interface between the client and the service must be known to both. It is important to note that typically, it is neither necessary, nor even desirable, for the client to know how the service will be performed. This is the same "loose coupling" of components that has been at the heart of good single-source application programs for many years. It is axiomatic for any SOA design. When adhered to, it has been shown to be capable of producing extraordinarily powerful and flexible applications systems.

Systems built in the spirit of SOA, whether consciously or not, are among the most successful in today's market. Google, Salesforce, and Amazon are just a few well-known examples of this. Most market-leading application software providers have publicly committed to adopting the SOA design model by the end of this decade.

This raises two vital questions. How can organizations with huge investments in legacy applications migrate to an SOA-based architecture, and can they do so within practical budgets and quickly enough to be competitive?

The Choices for COBOL, Until Now

In the past, the choices for modernizing and/or re-hosting applications written in COBOL have been less than ideal. Before the advent of practical web technology, the modernization of COBOL focused on two main problems: how to make the application "look" better, and how to mitigate the notion that COBOL was a "closed system" for development. The solutions to these problems were achieved by expanding both the number and kind of functions that could be performed within the COBOL environment. This was often done by expanding the already bloated COBOL

language to be more expressive with respect to newer programming methods and operating environments. As COBOL becomes less and less like the familiar language of the past, and more and more like the other object-oriented programming languages, it often becomes less accessible to the existing programming staff. Often the only good reasons for continuing to use COBOL are rendered moot. Now, however, there is a much better alternative for the organization with a COBOL legacy: the Web.

The Web is not just an Internet browsing platform. Today it is a constellation of technologies and standards that, along with the Internet, enable truly distributed applications to be built. Additionally, the Web's ubiquity means that virtually every computer-literate worker is not only comfortable with the graphical user interface (GUI) presented by Web browsers, they often are more familiar with it than with the native GUI on the desktop platform.

All the ingredients are now in place for the next generation of IT solutions, but is COBOL left out?

The Xcentrisity™ Solution

To have a realistic chance of relevance, applications written in COBOL must be able to 1) use the browser GUI, 2) provide business logic that can be called upon by not just the user, but also by non-COBOL resources, and 3) retain the familiar data-centric program orientation that is central to every

legacy business object and that is COBOL's primary strength. In short, COBOL must be able to play well with others and retain its value as a productive tool for the legacy COBOL practitioner.

At least one long-time supplier of COBOL language compilers, tools and application platforms, Liant Software Corporation, has targeted precisely this problem. Since the beginning of this millennium, Liant has recognized the technology current flowing in this direction, and has formulated the optimum tool set for this unique COBOL challenge.

Liant's answer to this problem, and its real-world constraints, is the Xcentrisity family of solution-building tools and technologies. Xcentrisity tools are designed to the central requirements of a Web-based IT resource. They are built around the key technologies of eXtensible Markup Language (XML) and HyperText Transfer Protocol (HTTP). And, most importantly, they mate these technologies with legacy COBOL business logic in a way that is straightforward and does not undermine the value of preserved and reused IT functions. Furthermore, when used in conjunction with a non-COBOL application framework, Xcentrisity allows a pure SOA system to be built and maintained around mature, well-understood COBOL business services.

Faster, Less Expensive, Better

To produce an SOA system with COBOL services, only the legacy user input/output logic need be replaced. Virtually all of the COBOL code implementing the core business objects can be reused, after removing or deactivating the direct user interface logic. This means that the SOA application can be built without redesign and reimplementing of those core functions. This greatly focuses the new code design

and development, and leads to a system with critical components that have, in most cases, decades of maturity and the reliability that results. Typically, a three- to five-year development cycle can be reduced to less than a year, assuming that it is practical to reuse legacy COBOL components.

Leverages Legacy Skills and Knowledge

When an SOA application is built around Xcentrisity components, the new COBOL programming that must be done does not require an extensive learning or retraining curve for the COBOL programmers and designers. The COBOL programmers' view of the Web world is seen through a lens that is natural, and even sometimes obvious to them. As a result, virtually all of the investment in the programming staff over the past years is retained and leveraged.

Future-proof

Perhaps most important, the resulting application system is modern and maintainable in all respects. The services exposed by or embedded in the SOA application are indistinguishable from those implemented using non-COBOL languages. The user interface can harness all of the power available in today's or tomorrow's Web browsers, and the applications' persistent data can take the form of any type of database, file or data object desired. As technology advances and new capabilities become practical, the SOA application can take advantage of

such developments with no change to the basic architecture.

So, how does it work?

How to Implement an SOA Solution

The tools provided by Xcentrisity permit a wide range of Web application architectures to be used while still retaining and reusing most of the legacy COBOL code. The most powerful and robust architectures are, however, those adhering to the principles of SOA.

The pivotal characteristic of an SOA application is its reliance on loose coupling of users of services and the services themselves. This means that the user of a service (including a COBOL-based service) does not—in fact cannot—know anything about how the service is performed unless that knowledge is included in the interface itself. In other words, it is completely up to the service implementation to perform whatever work is needed in whatever way it wants in order to satisfy the request. This is the principle of implementation transparency.

For most COBOL-based IT applications, the key services in an SOA system correspond to basic business “objects” for the problem domain being addressed. These objects would normally correspond to files (or groups of related files) that model a business element, such as a customer, a sales representative, the general ledger, an order, and so on. Most, if not all of the business objects that services need to access and operate on are already present in a legacy COBOL application. The COBOL record area represents the instance data associated with each object. As a result, much of the application design and architecture is

already defined by the legacy COBOL program(s).

Identify the Business Objects

The first step in a COBOL SOA transformation is to identify those record areas representing business objects that need to be exposed to the user. This defines the first set of services (or *primary* services) to be provided, namely those that directly operate on the core business objects. The next step is to determine what operations need to be supported for each of the business objects.

Luckily, most business objects in this category have a common set of functions that must be supported. These functions are often referred to as the Create, Read, Uppdate, and Delete operations, or simply CRUD. Additionally, most business objects also need to provide a searching capability in order to locate a specific instance or list of instances satisfying certain criteria. This means that we can decide, for each of the business objects identified, which of the CRUD functions must be supported, and whether or not a search function is required. This forms the first set of services provided by COBOL.

The Xcentrisity Business Information Server (BIS) is used as the platform for the COBOL services. When run by BIS, the COBOL programs necessary to provide the CRUD services for each business object can be easily provided. Each record area, along with the functions to be provided on

the corresponding business object instance, are coded in a manner similar to a standard COBOL CALL interface.

Define the User Interface

At this point, the decision must be made as to what degree the application will rely on a pre-defined web application framework or, in some cases, whether it will be built from scratch using only the basic foundation provided by BIS. While the latter is sometimes the only choice, especially for very specific (and unusual) processes, in almost all cases it is most efficient and practical to adopt a framework.

A framework, in this case, is a set of web application software components that define the overall presentation and control flow of the online application, and provide many of the general-purpose mechanisms that the application will need. There are many framework packages available, both open source and proprietary. The choice of which one, or none, is determined by three things: 1) the web server to be used, 2) the web application server to be used, and 3) whether or not the goals of the application to be built can be met with a specific choice of framework.

Among the many choices of framework, there are two that deserve first consideration when the service objects are implemented in COBOL and served by Xcentrisity BIS. One of them is a product by England Technical Services (ETS) called BIS-Express, and another is an Xcentrisity product called Frameware. Both of these products use an architecture commonly known as Model-View-Controller, or MVC, and both interface easily and seamlessly with the BIS platform. They differ in the form of the user interface (the view + controller) and the details of the interface

with the COBOL business logic. Xcentrisity Framework provides, in addition to the framework elements, general-purpose end-user functionality that may save time and effort in constructing the application. For this white paper, we will assume the use of Framework for the construction of an SOA COBOL application.

An application built with Xcentrisity Framework will run on any web application server platform that supports the Sun Java 2 Platform, Enterprise Edition (J2EE) standards. The COBOL services used by the application (that is, the migrated business objects) can be run on any OS platform that supports either Microsoft IIS or the Apache HTTP server on which Xcentrisity BIS is available. This list currently includes Windows, Linux, SCO and Solaris. Of course, the J2EE server and the HTTP server can be run on the same system if desired.

For each COBOL business object there typically will be a small set of user interactions (or forms) required to allow the human user to control aspects of the object. Sometimes these forms will enable only searching and listing (or reporting) the object, but more often some or all of the CRUD functions will be supported in some way. Framework provides a pre-determined look and feel for the application, as well as a set of generic pre-defined forms to accomplish the most common of these CRUD/search operations on IT business objects. Many times an application can be built in its entirety using these pre-defined forms. If

that is the case, little or no J2EE programming will be required and, therefore, only relatively common web graphical design work will be needed in addition to the COBOL skills already present in your organization.

After choosing the forms that will allow the user to interface with your business objects, a straightforward declarative file defines the view aspect of the application.

Implement the Primary Services

At this point, the COBOL programmers can design and code the implementation of the services to be used by the application. In most cases, this consists of collecting or exposing COBOL code that already exists in the legacy application, and providing the “glue” code to mate with the SOAP requirements. All of this programming is done in the familiar COBOL language.

Identify and Implement the Secondary Services

After determining the business objects that directly interact with the user, the remaining business objects represent those that are manipulated either as a result of the business process triggered by one or more of the user interactions (a *secondary* service) or only as a consequence of the work done to serve another business object. The secondary services will need to be exposed as web services, so that the business process flow implemented by Framework’s built-in workflow engine can call upon them at the appropriate time.

Secondary services differ from primary services in that they have no direct user interface. As a result, the choice of services for each business object is completely

determined by the nature of work required of the service. This often includes the CRUD functions that most of the primary services must implement, but it also often includes other, business domain-specific, functions.

The good news is that the legacy application has already determined what these services are, so the SOA designer need only provide the SOAP aware interface to them.

Define the Workflow, Test and Refine the Application

At this point, the overall structure of the SOA application is complete. All that remains is to use graphical workflow tools to define the data flow for the particular business domain being addressed and test and refine the services.

Summary

With the modern foundation of a J2EE web application server, Liant's Xcentrisity Framework and BIS, and the knowledge and skill of your own COBOL programming staff, a complete, modern, and well-architected SOA application system can be built. Furthermore, it can be built for a reasonable cost, both in terms of labor and in terms of calendar time, resulting in a good return-on-investment compared to other approaches. Finally, the system will be poised for further enhancements and adaptations, as business needs change and the body of web-based application technology progresses.

ⁱ Reference Model for Service Oriented Architecture 1.0, OASIS, Public Review Draft 2, 31 May 2006.